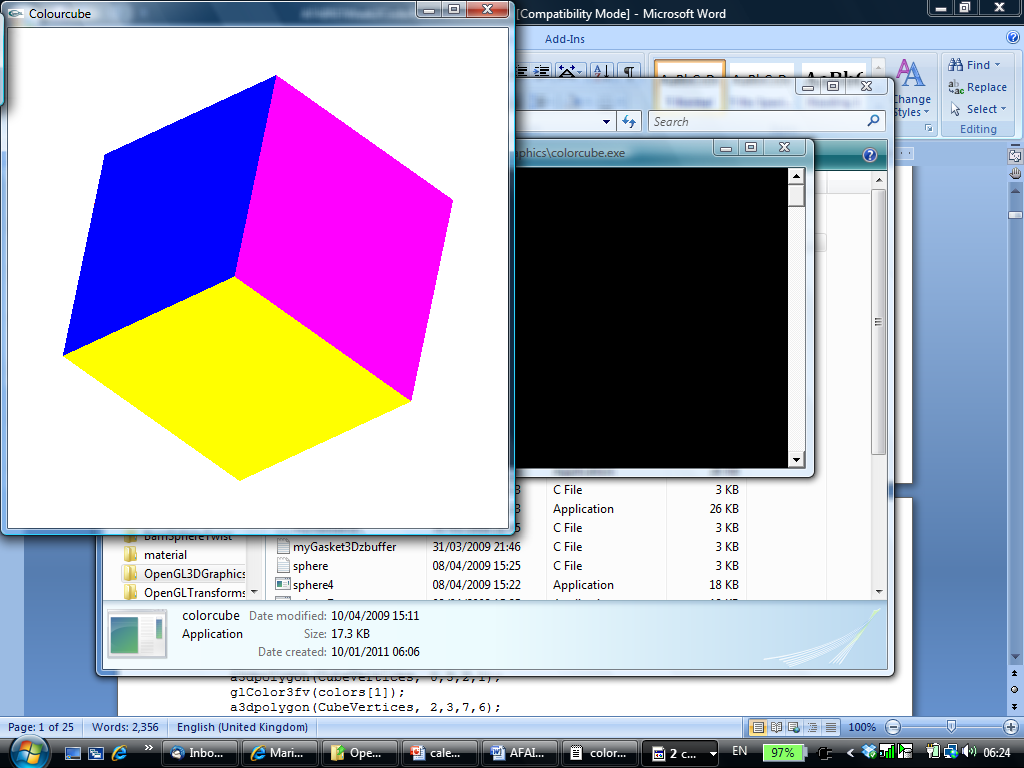
EBU5405 – week 2 – Code & Exercises

1. **colourCube.c**



/\* demonstration of use of homogeneous-coordinate transformations

and simple data structure for representing cube \*/

/\*cube is centered at origin \*/

#include <stdlib.h>

#include <GL/glut.h>

GLfloat X = 0.5; /\* A scaling factor \*/

static GLfloat theta[] = {45.0,45.0,45.0};

static GLint axis = 2;

GLfloat GlobalVertices[][3] = {{-1.0,-1.0,1.0},

{-1.0,1.0,1.0}, {1.0,1.0,1.0}, {1.0,-1.0,1.0},

{-1.0,-1.0,-1.0}, {-1.0,1.0,-1.0}, {1.0,1.0,-1.0},

{1.0,-1.0,-1.0}};

// These will be the coordinates of the vertices of the cube

GLfloat CubeVertices[][3] = {{-1.0,-1.0,1.0},

{-1.0,1.0,1.0}, {1.0,1.0,1.0}, {1.0,-1.0,1.0},

{-1.0,-1.0,-1.0}, {-1.0,1.0,-1.0}, {1.0,1.0,-1.0},

{1.0,-1.0,-1.0}};

GLfloat colors[][3] = {{0.0,0.0,0.0},{0.0,1.0,0.0},

{1.0,0.0,1.0}, {1.0,0.0,0.0}, {0.0,0.0,1.0},

{1.0,1.0,0.0}};

void a3dpolygon(GLfloat vertices[][3], int a, int b, int c, int d) {

/\* draw a polygon via list of vertices \*/

glShadeModel(GL\_FLAT);

glBegin(GL\_POLYGON);

glVertex3fv(vertices[a]);

glVertex3fv(vertices[b]);

glVertex3fv(vertices[c]);

glVertex3fv(vertices[d]);

glEnd();

}

void colorcube()

{

/\* map vertices to facets \*/

glColor3fv(colors[0]);

a3dpolygon(CubeVertices, 0,3,2,1);

glColor3fv(colors[1]);

a3dpolygon(CubeVertices, 2,3,7,6);

glColor3fv(colors[2]);

a3dpolygon(CubeVertices, 3,0,4,7);

glColor3fv(colors[3]);

a3dpolygon(CubeVertices, 1,2,6,5);

glColor3fv(colors[4]);

a3dpolygon(CubeVertices, 4,5,6,7);

glColor3fv(colors[5]);

a3dpolygon(CubeVertices, 5,4,0,1);

}

void display()

{

/\* display callback, clear frame buffer and z buffer,

rotate cube and draw, swap buffers \*/

glClear(GL\_COLOR\_BUFFER\_BIT | GL\_DEPTH\_BUFFER\_BIT);

glMatrixMode(GL\_MODELVIEW);

glLoadIdentity();

glRotatef(theta[0], 1.0, 0.0, 0.0);

glRotatef(theta[1], 0.0, 1.0, 0.0);

glRotatef(theta[2], 0.0, 0.0, 1.0);

colorcube();

glutSwapBuffers();

}

void spinCube()

{

/\* idle callback, spin cube about selected axis \*/

theta[axis] += 0.01;

if( theta[axis] > 360.0 ) theta[axis] -= 360.0;

glutPostRedisplay();

}

void specialkey(int key, int x, int y) {

switch (key) {

case GLUT\_KEY\_LEFT:

axis = 0;

break;

case GLUT\_KEY\_UP:

axis = 1;

break;

case GLUT\_KEY\_RIGHT:

axis = 2;

break;

}

}

void init()

{

int i, j;

glClearColor(1.0f, 1.0f, 1.0f, 1.0f);

for (j = 0; j < 3; j++) {

for (i = 0; i < 8; i++) {

CubeVertices[i][j] = GlobalVertices[i][j]\*X; //Scale each vertex by X

}

}

}

int main(int argc, char \*\*argv)

{

glutInit(&argc, argv);

/\* need both double buffering and z buffer \*/

glutInitDisplayMode(GLUT\_DOUBLE | GLUT\_RGB | GLUT\_DEPTH);

glutInitWindowSize(500, 500);

glutCreateWindow("Colourcube");

glutDisplayFunc(display);

glutIdleFunc(spinCube);

glutSpecialFunc(specialkey);

glEnable(GL\_DEPTH\_TEST); /\* Enable hidden-surface removal \*/

init();

glutMainLoop();

}

1. **barnWithWindows.c**

**Exercise 1 (code to be completed): decompose the barn into 16 faces and declare them**

/\* A rotating barn with 2 doors and a window \*/

#include <stdlib.h>

#include <GL/glut.h>

GLfloat X = 0.5; /\* A scaling factor \*/

static GLfloat theta[] = {0.0,0.0,0.0};

static GLint axis = 1;

GLfloat GlobalVertices[][3] = {{-1.0,-1.0,-1.5}, {1.0,-1.0,-1.5},

{1.0,1.0,-1.5}, {0.0,2.0,-1.5},

{-1.0,1.0,-1.5}, {-1.0,-1.0, 1.5},

{1.0,-1.0,1.5}, {1.0,1.0,1.5},

{0.0,2.0,1.5}, {-1.0,1.0,1.5},

//窗口

{-0.5, 1.0, 1.5}, {0.5, 1.0, 1.5},

{0.5, 0.2, 1.5}, {-0.5, 0.2, 1.5},

//

{-1.0, 0.2, 1.5}, {1.0, 0.2, 1.5},

//

{1.0, 0.2, 1.2}, {1.0, 0.2, 0.3},

{1.0, -1.0, 0.3}, {1.0, -1.0, 1.2},

//

{1.0, 0.2, -0.3}, {1.0, 0.2, -1.2},

{1.0, -1.0, -1.2}, {1.0, -1.0, -0.3},

{1.0, 0.2, -1.5}};

GLfloat BarnVertices[][3] = {{-1.0,-1.0,-1.5}, {1.0,-1.0,-1.5},

{1.0,1.0,-1.5}, {0.0,2.0,-1.5},

{-1.0,1.0,-1.5}, {-1.0,-1.0, 1.5},

{1.0,-1.0,1.5}, {1.0,1.0,1.5},

{0.0,2.0,1.5}, {-1.0,1.0,1.5},

{-0.5, 1.0, 1.5}, {0.5, 1.0, 1.5},

{0.5, 0.2, 1.5}, {-0.5, 0.2, 1.5},

{-1.0, 0.2, 1.5}, {1.0, 0.2, 1.5},

{1.0, 0.2, 1.2}, {1.0, 0.2, 0.3},

{1.0, -1.0, 0.3}, {1.0, -1.0, 1.2},

{1.0, 0.2, -0.3}, {1.0, 0.2, -1.2},

{1.0, -1.0, -1.2}, {1.0, -1.0, -0.3},

{1.0, 0.2, -1.5}};

GLint BarnFaces[][6] = {

//5个普通平面

{0,1,6,5,-1},

{0,1,2,3,4,-1},

{0,5,9,4,-1},

{4,3,8,9,-1},

{3,2,7,8,-1},

{8,9,7,-1},

{14,15,6,5,-1},

{9,10,13,14,-1},

{11,7,15,12,-1},

{7,2,24,15,-1},

{15,16,19,6,-1},

{17,20,23,18,-1},

{21,24,1,22,-1},

//窗户

{10,11,12,13,-1},

{16,17,18,19,-1},

{20,21,23,22,-1},

};

GLfloat colors[][3] = {{0.0,0.0,0.0},{1.0,0.0,0.0},{1.0,1.0,0.0},

{0.0,1.0,0.0}, {0.0,0.0,1.0},{1.0,0.0,1.0},

{0.5,0.5,0.5}, {0.0,1.0,1.0}, {1.0, 1.0, 1.0}};

void a3dpolygon(GLfloat vertices[][3], GLint face[]) {

/\* draw a polygon via list of vertices \*/

int i = 0;

int id;

glShadeModel(GL\_FLAT);

glBegin(GL\_POLYGON);

while (face[i] > -1) {

id = face[i];

glVertex3fv(vertices[id]);

i++;

}

glEnd();

}

void barn()

{

/\* map vertices to facets \*/

glColor3fv(colors[0]);

a3dpolygon(BarnVertices, BarnFaces[0]);

glColor3fv(colors[1]);

a3dpolygon(BarnVertices, BarnFaces[1]);

glColor3fv(colors[2]);

a3dpolygon(BarnVertices, BarnFaces[2]);

glColor3fv(colors[4]);

a3dpolygon(BarnVertices, BarnFaces[3]);

glColor3fv(colors[6]);

a3dpolygon(BarnVertices, BarnFaces[4]);

glColor3fv(colors[3]);

a3dpolygon(BarnVertices, BarnFaces[5]);

a3dpolygon(BarnVertices, BarnFaces[6]);

a3dpolygon(BarnVertices, BarnFaces[7]);

a3dpolygon(BarnVertices, BarnFaces[8]);

glColor3fv(colors[5]);

a3dpolygon(BarnVertices, BarnFaces[9]);

a3dpolygon(BarnVertices, BarnFaces[10]);

a3dpolygon(BarnVertices, BarnFaces[11]);

a3dpolygon(BarnVertices, BarnFaces[12]);

glColor3fv(colors[8]);

a3dpolygon(BarnVertices, BarnFaces[13]);

a3dpolygon(BarnVertices, BarnFaces[14]);

a3dpolygon(BarnVertices, BarnFaces[15]);

}

void display()

{

/\* display callback, clear frame buffer and z buffer \*/

glClear(GL\_COLOR\_BUFFER\_BIT | GL\_DEPTH\_BUFFER\_BIT);

glMatrixMode(GL\_MODELVIEW);

glLoadIdentity();

glRotatef(theta[0], 1.0, 0.0, 0.0);

glRotatef(theta[1], 0.0, 1.0, 0.0);

glRotatef(theta[2], 0.0, 0.0, 1.0);

barn();

glutSwapBuffers();

}

void spinBarn()

{

theta[axis] += 0.02;

if( theta[axis] > 360.0 ) theta[axis] -= 360.0;

glutPostRedisplay();

}

void specialkey(int key, int x, int y) {

switch (key) {

case GLUT\_KEY\_LEFT:

axis = 0;

break;

case GLUT\_KEY\_UP:

axis = 1;

break;

case GLUT\_KEY\_RIGHT:

axis = 2;

break;

}

}

void init()

{

int i, j;

glClearColor(1.0f, 1.0f, 1.0f, 1.0f);

for (i = 0; i < 25; i++) {

for (j = 0; j < 3; j++) {

BarnVertices[i][j] = GlobalVertices[i][j] \* X; // scaling

}

}

}

void myReshape(int w, int h)

{

glViewport(0, 0, w, h);

glMatrixMode(GL\_PROJECTION);

glLoadIdentity();

if (w <= h)

glOrtho(-1.5, 1.5, -1.5 \* (GLfloat) h / (GLfloat) w,

1.5 \* (GLfloat) h / (GLfloat) w, -10.0, 10.0);

else

glOrtho(-1.5 \* (GLfloat) w / (GLfloat) h,

1.5 \* (GLfloat) w / (GLfloat) h, -1.5, 1.5, -10.0, 10.0);

glMatrixMode(GL\_MODELVIEW);

glutPostRedisplay();

}

int main(int argc, char \*\*argv)

{

glutInit(&argc, argv);

/\* need both double buffering and z buffer \*/

glutInitDisplayMode(GLUT\_DOUBLE | GLUT\_RGB | GLUT\_DEPTH);

glutInitWindowSize(500, 500);

glutCreateWindow("A Barn");

glutDisplayFunc(display);

glutReshapeFunc(myReshape);

glutIdleFunc(spinBarn);

glutSpecialFunc(specialkey);

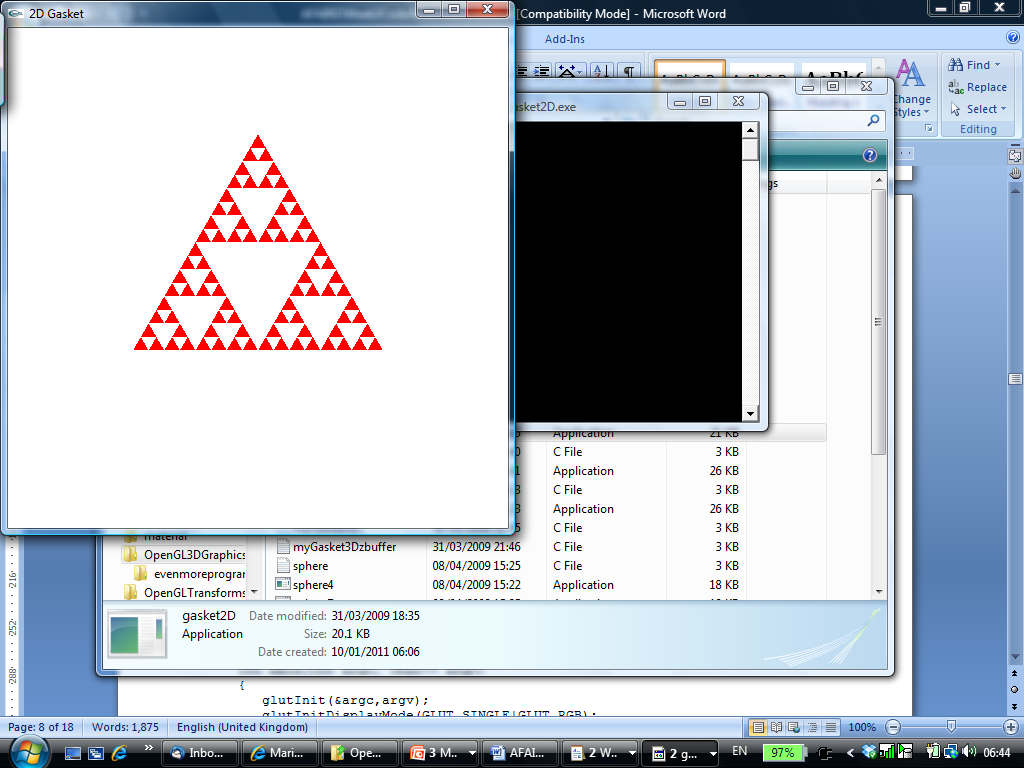
glEnable(GL\_DEPTH\_TEST); /\* Enable hidden-surface removal \*/

init();

glutMainLoop();

}

1. **SierpinskiGasket2D.c**



#include <GL/glut.h>

/\* initial triangle \*/

GLfloat v[3][2]={{-1.0, -0.58},{1.0, -0.58}, {0.0, 1.15}};

int n = 4; /\* number of recursive steps \*/

void triangle (GLfloat \*a, GLfloat \*b, GLfloat \*c)

{

/\* display one triangle \*/

glVertex2fv(a);

glVertex2fv(b);

glVertex2fv(c);

}

void divide\_triangle(GLfloat \*a, GLfloat \*b, GLfloat \*c, int m)

{

/\* triangle subdivision using vertex numbers \*/

GLfloat v0[2], v1[2], v2[2];

int j;

if(m>0)

{

for(j=0; j<2; j++) v0[j]=(a[j]+b[j])/2;

for(j=0; j<2; j++) v1[j]=(a[j]+c[j])/2;

for(j=0; j<2; j++) v2[j]=(b[j]+c[j])/2;

divide\_triangle(a, v0, v1, m-1);

divide\_triangle(c, v1, v2, m-1);

divide\_triangle(b, v2, v0, m-1);

}

else(triangle(a,b,c));

/\* draw triangle at end of recursion \*/

}

void display()

{

glClear(GL\_COLOR\_BUFFER\_BIT);

glBegin(GL\_TRIANGLES);

divide\_triangle(v[0], v[1], v[2], n);

glEnd();

glFlush();

}

void myinit()

{

glMatrixMode(GL\_PROJECTION);

glLoadIdentity();

gluOrtho2D(-2.0, 2.0, -2.0, 2.0);

glMatrixMode(GL\_MODELVIEW);

glClearColor (1.0, 1.0, 1.0, 1.0);

glColor3f(1.0,0.0,0.0);

}

int main(int argc, char \*\*argv)

{

glutInit(&argc, argv);

glutInitDisplayMode(GLUT\_SINGLE|GLUT\_RGB);

glutInitWindowSize(500, 500);

glutCreateWindow("2D Gasket");

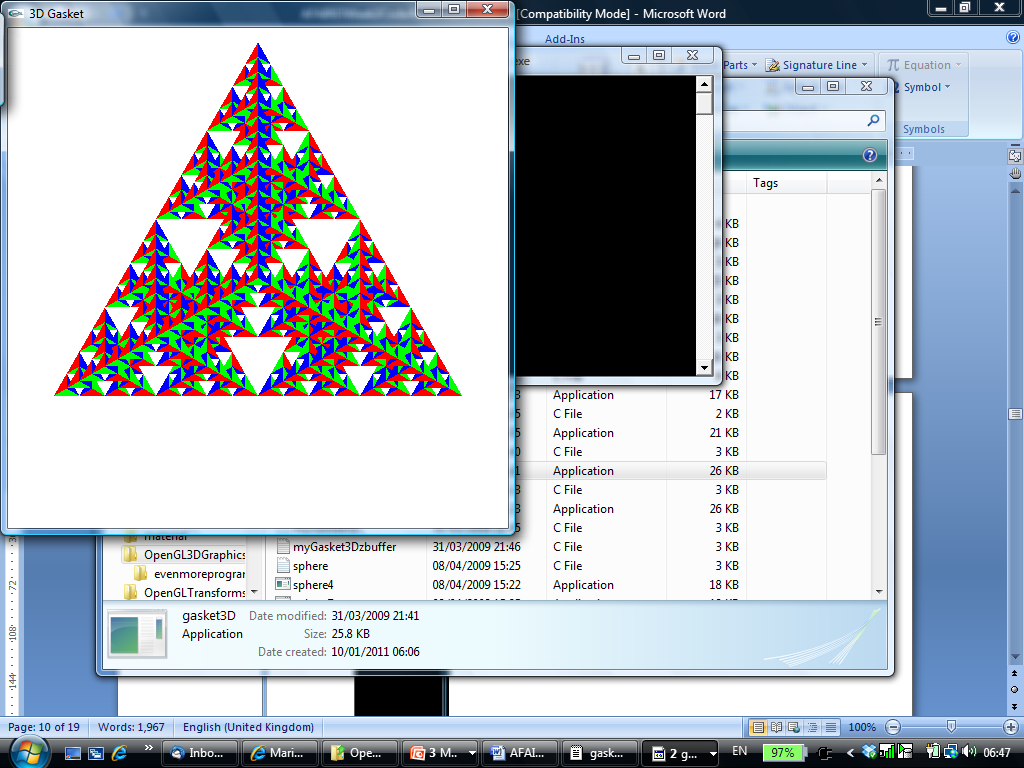
glutDisplayFunc(display);

myinit();

glutMainLoop();

}

1. **SierpinskiGasket3D.c**



#include <stdlib.h>

#include <GL/glut.h>

/\* initial tetrahedron \*/

GLfloat v[4][3]={{0.0, 0.0, 1.0}, {0.0, 0.942809, -0.33333},

{-0.816497, -0.471405, -0.333333},

{0.816497, -0.471405, -0.333333}};

GLfloat colors[4][3] = {{0.0, 0.0, 0.0}, {0.0, 0.0, 0.0},

{0.0, 0.0, 0.0}, {0.0, 0.0, 0.0}};

int n=4; /\* number of subdivision steps \*/

//ROTATE!!!!!!!!!!!!!!!!!!!!!

static GLfloat theta[] = {45.0,45.0,45.0};

static GLint axis = 2;

void triangle(GLfloat \*va, GLfloat \*vb, GLfloat \*vc)

{

glVertex3fv(va);

glVertex3fv(vb);

glVertex3fv(vc);

}

void tetra(GLfloat \*a, GLfloat \*b, GLfloat \*c, GLfloat \*d)

{

glColor3fv(colors[0]);

triangle(a, b, c);

glColor3fv(colors[1]);

triangle(a, c, d);

glColor3fv(colors[2]);

triangle(a, d, b);

glColor3fv(colors[3]);

triangle(b, d, c);

}

void divide\_tetra(GLfloat \*a, GLfloat \*b, GLfloat \*c, GLfloat \*d, int m)

{

GLfloat mid[6][3];

int j;

if(m>0)

{

/\* compute six midpoints \*/

for(j=0; j<3; j++) mid[0][j]=(a[j]+b[j])/2;

for(j=0; j<3; j++) mid[1][j]=(a[j]+c[j])/2;

for(j=0; j<3; j++) mid[2][j]=(a[j]+d[j])/2;

for(j=0; j<3; j++) mid[3][j]=(b[j]+c[j])/2;

for(j=0; j<3; j++) mid[4][j]=(c[j]+d[j])/2;

for(j=0; j<3; j++) mid[5][j]=(b[j]+d[j])/2;

/\* create 4 tetrahedrons by subdivision \*/

divide\_tetra(a, mid[0], mid[1], mid[2], m-1);

divide\_tetra(mid[0], b, mid[3], mid[5], m-1);

divide\_tetra(mid[1], mid[3], c, mid[4], m-1);

divide\_tetra(mid[2], mid[4], d, mid[5], m-1);

}

else(tetra(a,b,c,d)); /\* draw tetrahedron at end of recursion \*/

}

void display()

{

glClear(GL\_COLOR\_BUFFER\_BIT | GL\_DEPTH\_BUFFER\_BIT);

//ROTATE!!!!!!!!!!!!!

glMatrixMode(GL\_MODELVIEW);

glLoadIdentity();

glRotatef(theta[0], 1.0, 0.0, 0.0);

glRotatef(theta[1], 0.0, 1.0, 0.0);

glRotatef(theta[2], 0.0, 0.0, 1.0);

glBegin(GL\_TRIANGLES);

divide\_tetra(v[0], v[1], v[2], v[3], n);

glEnd();

glutSwapBuffers();

}

//ROTATE!!!!!!!!!!!!!!!!!!!!!!!!!1

void spinGasket()

{

/\* idle callback, spin cube about selected axis \*/

theta[axis] += 1;

if( theta[axis] > 360.0 ) theta[axis] -= 360.0;

glutPostRedisplay();

}

void specialkey(int key, int x, int y) {

switch (key) {

case GLUT\_KEY\_LEFT:

axis = 0;

break;

case GLUT\_KEY\_UP:

axis = 1;

break;

case GLUT\_KEY\_RIGHT:

axis = 2;

break;

}

}

void myReshape(int w, int h)

{

glViewport(0, 0, w, h);

glMatrixMode(GL\_PROJECTION);

glLoadIdentity();

if (w <= h)

glOrtho(-1.0, 1.0, -1.0 \* (GLfloat) h / (GLfloat) w,

1.0 \* (GLfloat) h / (GLfloat) w, -10.0, 10.0);

else

glOrtho(-1.0 \* (GLfloat) w / (GLfloat) h,

1.0 \* (GLfloat) w / (GLfloat) h, -1.0, 1.0, -10.0, 10.0);

glMatrixMode(GL\_MODELVIEW);

glutPostRedisplay();

}

int main(int argc, char \*\*argv)

{

glutInit(&argc, argv);

glutInitDisplayMode(GLUT\_DOUBLE | GLUT\_RGB | GLUT\_DEPTH);

glutInitWindowSize(500, 500);

glutCreateWindow("3D Gasket");

glutReshapeFunc(myReshape);

glutDisplayFunc(display);

glutIdleFunc(spinGasket);

glutSpecialFunc(specialkey);

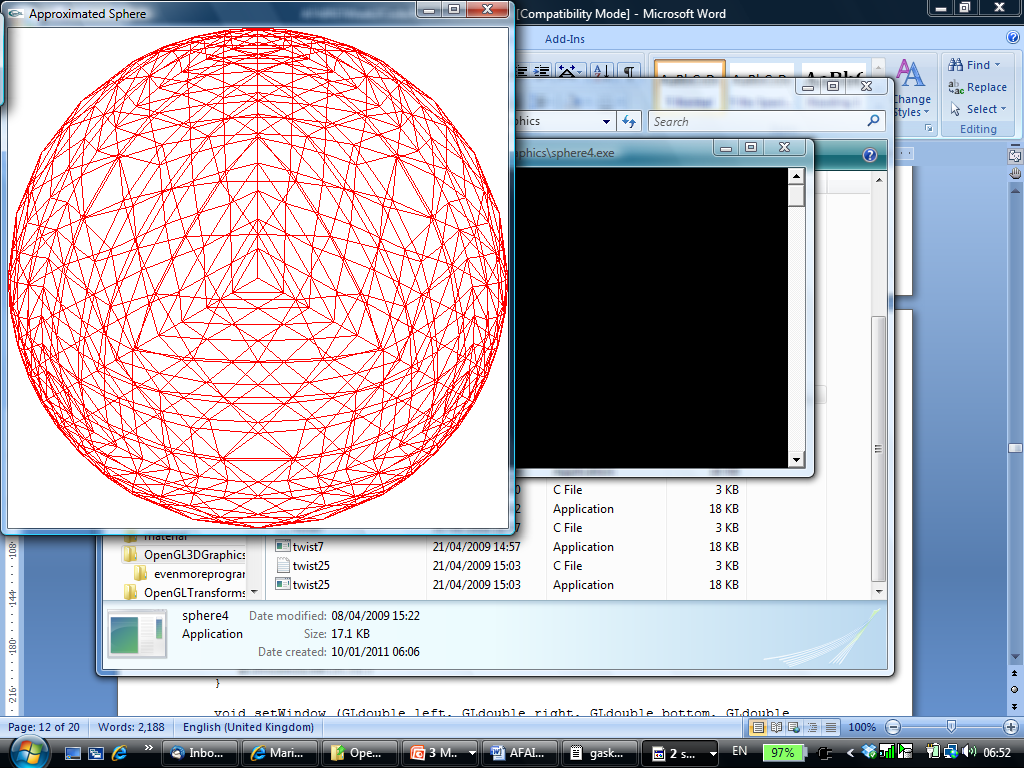
glEnable(GL\_DEPTH\_TEST);

glClearColor (1.0, 1.0, 1.0, 1.0);

glutMainLoop();

}}

1. **sphere.c**

****

**Exercise 2 (code to be completed): write the normalise function, which places every vertice on the surface of the sphere by normalising their distance to the center.**

/\* recursive subdivision of a tetrahedron to approximate a sphere \*/

#include <stdlib.h>

#include <GL/glut.h>

/\* initial tetrahedron \*/

GLfloat v[4][3]={{0.0, 0.0, 1.0}, {0.0, 0.942809, -0.33333},

{-0.816497, -0.471405, -0.333333},

{0.816497, -0.471405, -0.333333}};

int n;

void triangle(GLfloat \*va, GLfloat \*vb, GLfloat \*vc)

{

glBegin(GL\_LINE\_LOOP);

glVertex3fv(va);

glVertex3fv(vb);

glVertex3fv(vc);

glEnd();

}

// vertex normalisation

void normalise(GLfloat \*p)

{

double a = sqrt(p[0]\*p[0]+p[1]\*p[1]+p[2]\*p[2]);

int i;

for (i=0; i<3; i++)

{

p[i]=p[i]/a;

}

}

void divide\_triangle(GLfloat \*a, GLfloat \*b, GLfloat \*c, int n)

{

GLfloat v1[3], v2[3], v3[3];

int j;

if (n>0) {

for (j=0; j<3; j++) v1[j]=a[j]+b[j];

normalise(v1);

for (j=0; j<3; j++) v2[j]=a[j]+c[j];

normalise(v2);

for (j=0; j<3; j++) v3[j]=c[j]+b[j];

normalise(v3);

divide\_triangle(a, v2, v1, n-1);

divide\_triangle(c, v3, v2, n-1);

divide\_triangle(b, v1, v3, n-1);

divide\_triangle(v1, v2, v3, n-1);

}

else triangle (a, b, c);

}

void tetrahedron(int n)

{

divide\_triangle(v[0], v[1], v[2], n);

divide\_triangle(v[3], v[2], v[1], n);

divide\_triangle(v[0], v[3], v[1], n);

divide\_triangle(v[0], v[2], v[3], n);

}

void display(){

glClear(GL\_COLOR\_BUFFER\_BIT | GL\_DEPTH\_BUFFER\_BIT);

tetrahedron(n);

glFlush();

}

void myReshape(int w, int h){

glViewport(0, 0, w, h);

glMatrixMode(GL\_PROJECTION);

glLoadIdentity();

if (w <= h)

glOrtho(-1.0, 1.0, -1.0 \* (GLfloat) h / (GLfloat) w,

1.0 \* (GLfloat) h / (GLfloat) w, -10.0, 10.0);

else

glOrtho(-1.0 \* (GLfloat) w / (GLfloat) h,

1.0 \* (GLfloat) w / (GLfloat) h, -1.0, 1.0, -10.0, 10.0);

glMatrixMode(GL\_MODELVIEW);

glutPostRedisplay();

}

int main(int argc, char \*\*argv)

{

n=10; /\* number of subdivision steps \*/

glutInit(&argc, argv);

glutInitDisplayMode(GLUT\_SINGLE | GLUT\_RGB | GLUT\_DEPTH);

glutInitWindowSize(500, 500);

glutCreateWindow("Approximated Sphere");

glutReshapeFunc(myReshape);

glutDisplayFunc(display);

glEnable(GL\_DEPTH\_TEST);

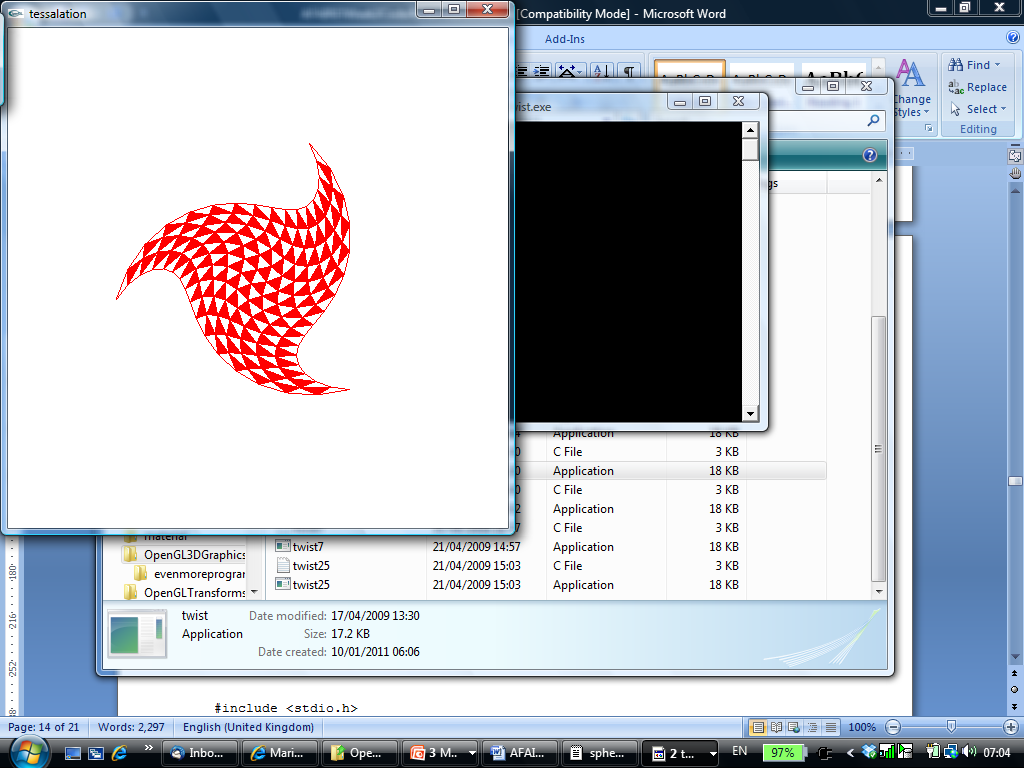
glClearColor (1.0, 1.0, 1.0, 1.0);

glColor3f(0.0, 0.0, 0.0);

glutMainLoop();

}

1. **twist.c**



**Exercise 3 (code to be completed): complete the triangle function in order to achieve a twist effect. The function should rotate each vertex by an angle that depends on the position of the vertex (calculated in d).**

#include <math.h>

#include <GL/glut.h>

/\* initial triangle \*/

GLfloat v[3][2]={{-1.0, -0.58}, {1.0, -0.58}, {0.0, 1.15}};

int n = 4; /\* number of recursive steps \*/

GLfloat twist = 1.5;

void triangle (GLfloat \*a, GLfloat \*b, GLfloat \*c)

{

/\* display one triangle \*/

GLfloat v[2];

double d;

glBegin(GL\_POLYGON);

d = sqrt(a[0]\*a[0] + a[1]\*a[1]);//距离啊距离！！

//第一个点！！！！

v[0] = a[0]\*cos(d\*twist)-a[1]\*sin(d\*twist);

v[1] = a[0]\*sin(d\*twist)+a[1]\*cos(d\*twist);

glVertex2fv(v);

d = sqrt(b[0]\*b[0] + b[1]\*b[1]);

//第二个点！！！！

v[0] = b[0]\*cos(d\*twist)-b[1]\*sin(d\*twist);

v[1] = b[0]\*sin(d\*twist)+b[1]\*cos(d\*twist);

glVertex2fv(v);

d = sqrt(c[0]\*c[0] + c[1]\*c[1]);

//第三个点！！！！ ！！！！！！！！！

v[0] = c[0]\*cos(d\*twist)-c[1]\*sin(d\*twist);

v[1] = c[0]\*sin(d\*twist)+c[1]\*cos(d\*twist);

glVertex2fv(v);

glEnd();

}

void divide\_triangle(GLfloat \*a, GLfloat \*b, GLfloat \*c, int m)

{

/\* triangle subdivision using vertex numbers \*/

GLfloat v[3][2];

int j;

if(m>0)

{

for(j=0; j<2; j++) v[0][j]=(a[j]+b[j])/2;

for(j=0; j<2; j++) v[1][j]=(a[j]+c[j])/2;

for(j=0; j<2; j++) v[2][j]=(b[j]+c[j])/2;

divide\_triangle(a, v[0], v[1], m-1);

divide\_triangle(v[0], b, v[2], m-1);

divide\_triangle(v[1], v[2], c, m-1);

divide\_triangle(v[0], v[1], v[2], m-1);

}

else(triangle(a,b,c));

/\* draw triangle at end of recursion \*/

}

void display()

{

glClear(GL\_COLOR\_BUFFER\_BIT);

divide\_triangle(v[0], v[1], v[2], n);

glFlush();

}

void myinit()

{

glMatrixMode(GL\_PROJECTION);

glLoadIdentity();

gluOrtho2D(-2.0, 2.0, -2.0, 2.0);

glMatrixMode(GL\_MODELVIEW);

glClearColor (1.0, 1.0, 1.0, 1.0);

glColor3f(1.0,0.0,0.0);

glPolygonMode(GL\_FRONT, GL\_LINE);

}

int main(int argc, char \*\*argv)

{

glutInit(&argc, argv);

glutInitDisplayMode(GLUT\_SINGLE|GLUT\_RGB);

glutInitWindowSize(500, 500);

glutCreateWindow("tessalation");

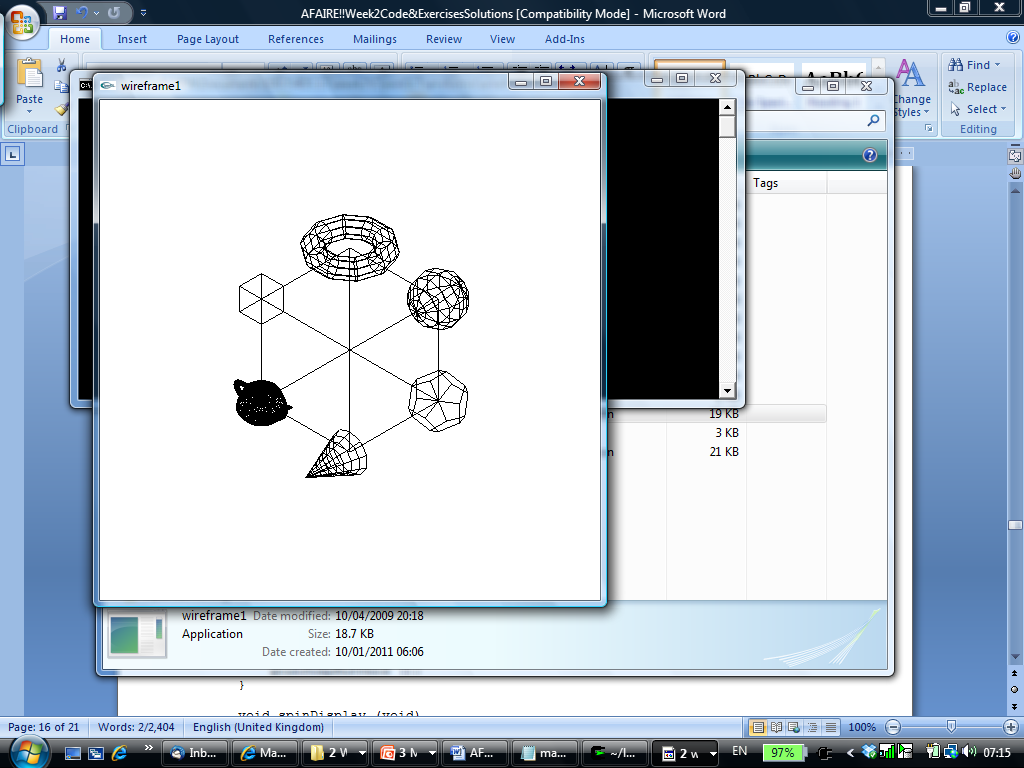
glutDisplayFunc(display);

myinit();

glutMainLoop();

}

1. **wireFrame.c**



#include <stdlib.h>

#include <GL/glut.h>

void display()

{

glMatrixMode(GL\_MODELVIEW); // position and aim the camera

glLoadIdentity();

gluLookAt(2.0, 2.0, 2.0, 0.0, 0.0, 0.0, 0.0, 1.0, 0.0);

glClear(GL\_COLOR\_BUFFER\_BIT); // clear the screen

glPushMatrix();

glTranslated(0.5, 0.5, 0.5); // big cube at (0.5, 0.5, 0.5)

glutWireCube(1.0);

glPopMatrix();

glPushMatrix();

glTranslated(1.0,1.0,0.0); // sphere at (1,1,0)

glutWireSphere(0.25, 10, 8);

glPopMatrix();

glPushMatrix();

glTranslated(1.0,0.0,1.0); // cone at (1,0,1)

glutWireCone(0.2, 0.5, 10, 8);

glPopMatrix();

glPushMatrix();

glTranslated(0.0,0.0,1.0);

glutWireTeapot(0.2); // teapot at (1,1,1)

glPopMatrix();

glPushMatrix();

glTranslated(0.0,1.0 ,0.0); // torus at (0,1,0)

glRotated(90.0, 1,0,0);

glutWireTorus(0.1, 0.3, 10, 10);

glPopMatrix();

glPushMatrix();

glTranslated(1.0,0.0,0.0); // dodecahedron at (1,0,0)

glScaled(0.15, 0.15, 0.15);

glutWireDodecahedron();

glPopMatrix();

glPushMatrix();

glTranslated(0.0,1.0,1.0); // small cube at (0,1,1)

glutWireCube(0.25);

glPopMatrix();

glFlush();

}

void reshape(int w, int h)

{

glViewport (0, 0, w, h);

glMatrixMode(GL\_PROJECTION); // set the view volume shape

glLoadIdentity();

glOrtho(-2.0, 2.0, -2.0, 2.0, -2.0, 100.0);

}

void init()

{

glClearColor(1.0, 1.0, 1.0, 1.0);

glColor3f(0.0, 0.0, 0.0);

}

int main(int argc, char \*\*argv)

{

glutInit(&argc, argv);

glutInitDisplayMode(GLUT\_SINGLE | GLUT\_RGB );

glutInitWindowSize(500,500);

glutInitWindowPosition(100, 100);

glutCreateWindow("wireframe");

glutDisplayFunc(display);

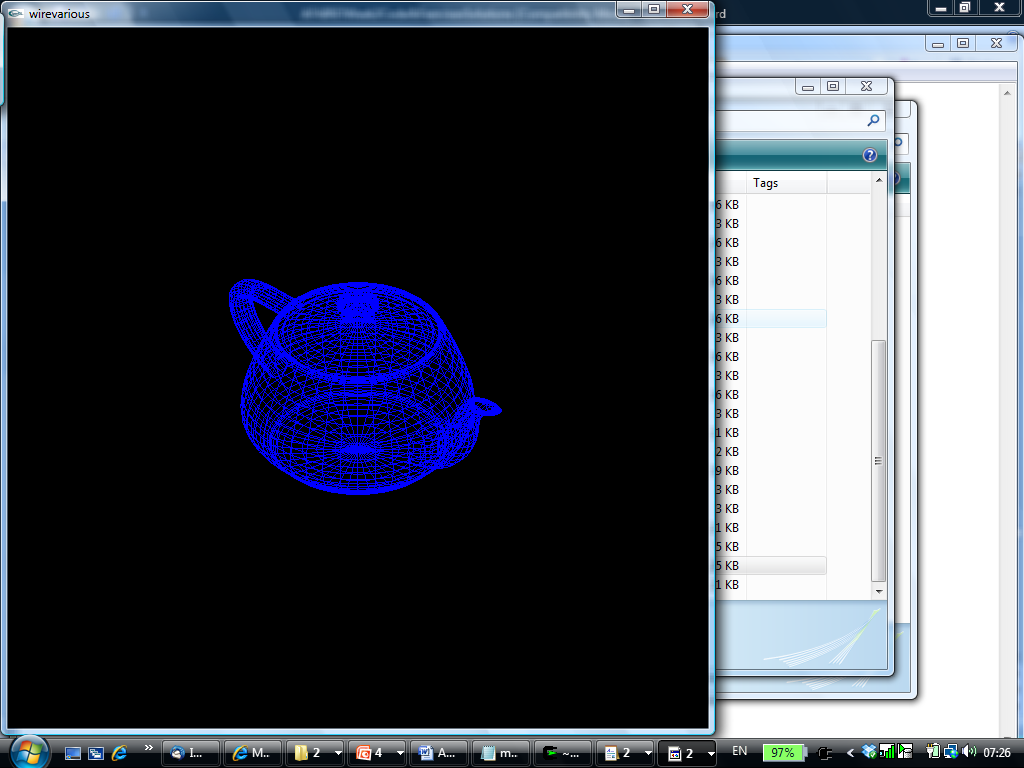
glutReshapeFunc(reshape);

init();

glutMainLoop();

}

1. **gluAndGlutObjects.c**



/\* This program displays a number of GLUT objects. Use the N and P keys to move to Next or Previous object. Click the right mouse button to start or stop the spin of the object. Click the left mouse button to change the viewpoint on the object (gluLookAt function parameters). \*/

#include <stdlib.h>

#include <GL/glut.h>

#define MAX 14

#define MIN 1

int next = MIN;

GLUquadricObj \*obj;

float eyex, eyey, eyez;

float atx, aty, atz;

int count = 0;

GLint eye[3] = {1, 0, 0};

GLboolean pause = GL\_TRUE;

void myInit(void) {

glClearColor(0.0, 0.0, 0.0, 0.0);

glColor3f(1.0, 1.0, 1.0);

eyex = eyey = eyez = 1.0;

atx = aty = atz = 0.0;

glEnable(GL\_DEPTH\_TEST);

}

void reshape(int w, int h) {

glViewport(0, 0, w, h);

glMatrixMode(GL\_PROJECTION);

glLoadIdentity();

glOrtho(-1.5, 1.5, -1.5, 1.5, -10.0, 10.0);

}

void mykey(unsigned char key, int x, int y)

{

if(key == 'N' | key == 'n')

next++;

if(key == 'P' | key == 'p')

next--;

if (next > MAX) next = MIN;

if (next < MIN) next = MAX;

if (next <= 5) glColor3f(1.0, 1.0, 1.0);

if (next > 5 && next <= 9) glColor3f(1.0, 1.0, 0.0);

if (next > 9 && next <= 13) glColor3f(1.0, 0.0, 0.0);

if (next == 14) glColor3f(0.0, 0.0, 1.0);

glutPostRedisplay();

}

void mydisplay(){

glClear(GL\_COLOR\_BUFFER\_BIT | GL\_DEPTH\_BUFFER\_BIT);

glMatrixMode(GL\_MODELVIEW);

glLoadIdentity();

gluLookAt(eyex, eyey, eyez, atx, aty, atz, 0.0, 1.0, 0.0);

obj = gluNewQuadric();

gluQuadricDrawStyle(obj, GLU\_LINE);

switch(next) {

case 1:

gluSphere(obj, 0.5, 20, 20);

break;

case 2:

gluCylinder(obj, 0.5, 0.3, 1, 10, 10);

break;

case 3:

gluDisk(obj, 0.2, 0.5, 10, 10);

break;

case 4:

gluDisk(obj, 0.2, 0.5, 30, 30);

break;

case 5:

gluPartialDisk(obj, 0.2, 0.5, 30, 30, 0, 45);

break;

case 6:

glutWireSphere(0.5, 20, 20);

break;

case 7:

glutSolidSphere(0.5, 20, 20);

break;

case 8:

glutWireCone(0.5, 1, 10, 10);

break;

case 9:

glutWireTorus(0.2, 0.5, 20, 20);

break;

case 10:

glutWireCube(1);

break;

case 11:

glutWireTetrahedron();

break;

case 12:

glutWireOctahedron();

break;

case 13:

glutWireIcosahedron();

break;

case 14:

glutWireTeapot(0.5);

break;

}

glutSwapBuffers();

}

void mymouse(GLint button, GLint state, GLint x, GLint y) {

if (button==GLUT\_LEFT\_BUTTON && state==GLUT\_DOWN) {

if (eye[0] == 1) {

eye[0] = 0;

eye[1] = 1;

return;

}

if (eye[1] == 1) {

eye[1] = 0;

eye[2] = 1;

return;

}

if (eye[2] == 1) {

eye[2] = 0;

eye[0] = 1;

return;

}

}

if (button==GLUT\_RIGHT\_BUTTON && state==GLUT\_DOWN) {

pause = !pause;

}

}

void idle() {

if (pause) return;

if (eye[0] == 1) {

eyex += 0.001;

if (eyex > 1.0) eyex = -1.0;

}

if (eye[1] == 1) {

eyey += 0.001;

if (eyey > 1.0) eyey = -1.0;

}

if (eye[2] == 1) {

eyez += 0.001;

if (eyez > 1.0) eyez = -1.0;

}

glutPostRedisplay();

}

int main(int argc, char\*\* argv){

glutInit(&argc, argv);

glutInitDisplayMode(GLUT\_DOUBLE | GLUT\_RGB | GLUT\_DEPTH);

glutInitWindowSize(700, 700);

glutCreateWindow("GluAndGlutObjects");

glutKeyboardFunc(mykey);

glutMouseFunc(mymouse);

glutDisplayFunc(mydisplay);

glutReshapeFunc(reshape);

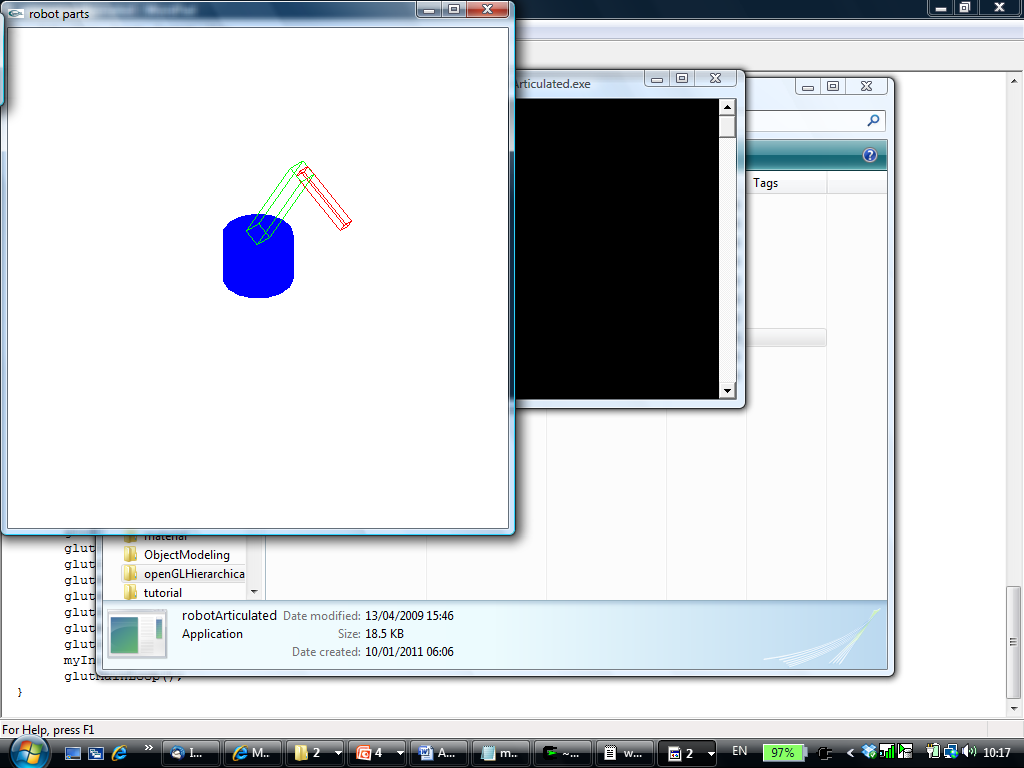
glutIdleFunc(idle);

myInit();

glutMainLoop();

}

1. **robotArm.c**



/\*robot arm three parts animated \*/

/\* rotate base : press b (to stop, press b again) \*/

/\* rotate lower arm : press l (to stop, press l again) \*/

/\* rotate upper arm : press u (to stop, press u again) \*/

#include <stdlib.h>

#include <GL/glut.h>

#define BASE\_RADIUS 1.0

#define BASE\_HEIGHT 1.5

#define LOWER\_ARM\_WIDTH 0.5

#define LOWER\_ARM\_HEIGHT 3.5

#define UPPER\_ARM\_WIDTH 0.3

#define UPPER\_ARM\_HEIGHT 2.0

#define HAND\_RADIUS 0.5

#define HAND\_HEIGHT 0.5

GLfloat theta[4] = {0.0, 0.0, 0.0,0.0};

GLboolean bas = GL\_FALSE;

GLboolean low = GL\_FALSE;

GLboolean up = GL\_FALSE;

GLboolean han = GL\_FALSE;

GLint u\_direction = 1;

GLint l\_direction = 1;

GLint h\_direction = 1;

GLUquadricObj \*p; // pointer to quadric object????

void base()

{

glPushMatrix();

glRotatef (-90.0, 1.0, 0.0, 0.0);

gluCylinder (p, BASE\_RADIUS, BASE\_RADIUS, BASE\_HEIGHT, 5, 5);

glPopMatrix();

}

void lower\_arm()

{

glPushMatrix();

glTranslatef(0.0, 0.5\*LOWER\_ARM\_HEIGHT, 0.0);

glScalef(LOWER\_ARM\_WIDTH, LOWER\_ARM\_HEIGHT, LOWER\_ARM\_WIDTH);

glutWireCube(1.0);

glPopMatrix();

}

void upper\_arm()

{

glPushMatrix();

glTranslatef(0.0, 0.5\*UPPER\_ARM\_HEIGHT, 0.0);

glScalef(UPPER\_ARM\_WIDTH, UPPER\_ARM\_HEIGHT, UPPER\_ARM\_WIDTH);

glutWireCube(1.0);

glPopMatrix();

}

void hand()

{

glPushMatrix();

glTranslatef(0.0, 0.5\*HAND\_HEIGHT, 0.0);

glRotatef (-90.0, 1.0, 0.0, 0.0);

gluCylinder (p, HAND\_RADIUS, HAND\_RADIUS, HAND\_HEIGHT, 5, 5);

glPopMatrix();

}

void display()

{

glClear(GL\_COLOR\_BUFFER\_BIT | GL\_DEPTH\_BUFFER\_BIT);

glMatrixMode(GL\_MODELVIEW);

glLoadIdentity();

gluLookAt(1.0, 1.0, 1.0, 0.0, 0.0, 0.0, 0.0, 1.0, 0.0);

glRotatef(theta[0], 0.0, 1.0, 0.0);//大家一起转

glColor3f (0.0, 0.0, 1.0);

base();

glTranslatef(0.0, BASE\_HEIGHT, 0.0);//将坐标上移base height

glRotatef(theta[1], 0.0, 0.0, 1.0);//上面的大家一起转

glColor3f (0.0, 1.0, 0.0);

lower\_arm();

glTranslatef(0.0, LOWER\_ARM\_HEIGHT, 0.0);//将坐标上移arm height

glRotatef(theta[2], 0.0, 0.0, 1.0);

glColor3f (1.0, 0.0, 0.0);

upper\_arm();

glTranslatef(0.0, UPPER\_ARM\_HEIGHT, 0.0);

glRotatef(theta[3], 0.0, 0.0, 1.0);

glColor3f (0.0, 0.0, 0.0);

hand();

glutSwapBuffers();

}

void init()

{

glClearColor(1.0f, 1.0f, 1.0f, 1.0f);

p = gluNewQuadric();//??????

}

void reshape(int w, int h)

{

glViewport (0, 0, w, h);

glMatrixMode(GL\_PROJECTION);

glLoadIdentity();

glOrtho(-7.0, 7.0, -7.0, 7.0, -10.0, 10.0);

}

void mykey(unsigned char key, int x, int y)

{

if(key == 'b' | key == 'B')

bas = !bas;

if(key == 'u' | key == 'U')

up = !up;

if(key == 'l' | key == 'L')

low = !low;

if(key == 'h' | key == 'H')

han = !han;

}

void idle()

{

if (bas) {

theta[0] += 0.2;

if (theta[0] > 360) theta[0] -= 360;

}

///base

if (low) {

if (l\_direction > 0) {

theta[1] += 0.2;

if (theta[1] > 70) {

l\_direction = -1;

theta[1] -= 0.2;

}

}

else {

theta[1] -= 0.2;

if (theta[1] < -70) {

l\_direction = 1;

theta[1] += 0.2;

}

}

}

//low!!!!!

if (up) {

if (u\_direction > 0) {

theta[2] += 0.2;

if (theta[2] > 70) {

u\_direction = -1;

theta[2] -= 0.2;

}

}

else {

theta[2] -= 0.2;

if (theta[2] < -70) {

u\_direction = 1;

theta[2] += 0.2;

}

}

}

////up!!!!

if (han) {

if (h\_direction > 0) {

theta[3] += 0.2;

if (theta[3] > 70) {

h\_direction = -1;

theta[3] -= 0.2;

}

}

else {

theta[3] -= 0.2;

if (theta[3] < -70) {

h\_direction = 1;

theta[3] += 0.2;

}

}

}

///hand!!!!!!

glutPostRedisplay();

}

int main(int argc, char \*\*argv)

{

glutInit(&argc, argv);

glutInitDisplayMode(GLUT\_DOUBLE | GLUT\_RGB | GLUT\_DEPTH);

glutInitWindowSize(500, 500);

glutCreateWindow("robot animated");

glutKeyboardFunc(mykey);

glutIdleFunc(idle);

glutDisplayFunc(display);

glutReshapeFunc(reshape);

glEnable(GL\_DEPTH\_TEST); /\* Enable hidden-surface removal \*/

init();

glutMainLoop();

}